Due Friday, April 14 in class

1. (20 points) The goal is to write a routine to find roots of a polynomial using Rayleigh quotient iteration. You may write your program in Matlab, C, or Python. You should do exception handling only to the extent necessary to complete the requested computations. Solving a linear system with a singular matrix is an exception that you should be able to recover from!

(a) Write a routine which solves a linear system of the form

\[
\begin{bmatrix}
  a_1 & d_1 \\
  b_2 & a_2 & d_2 \\
  \ddots & \ddots & \ddots \\
  b_{n-1} & a_{n-1} & d_{n-1} & x_{n-1} \\
  b_n & a_n & d_n & x_n
\end{bmatrix}
\begin{bmatrix}
  x_1 \\
  x_2 \\
  \vdots \\
  x_{n-1} \\
  x_n
\end{bmatrix}
= \begin{bmatrix}
  f_1 \\
  f_2 \\
  \vdots \\
  f_{n-1} \\
  f_n
\end{bmatrix}.
\]

The routine should have as input arrays \( a \) and \( b \) of dimension \( n - 1 \) and arrays \( d \) and \( f \) of dimension \( n \). On output the elements of \( f \) should be overwritten with the solution. The elements of \( a \) and \( d \) will also be changed. Use Gaussian elimination with partial pivoting and one extra array of working storage of length \( n \). (This extra array of storage is unnecessary but will produce more readable code.)

(b) Write a routine that finds a root of a polynomial

\[ c_n x^n + c_{n-1} x^{n-1} + \cdots + c_1 x + c_0, \quad c_n \neq 0, \]

near some specified value \( r \) by (i) forming the companion matrix multiplied by \( c_n \), i.e.,

\[
\begin{bmatrix}
  0 & -c_0 \\
  c_n & 0 & -c_1 \\
  \ddots & \ddots & \ddots \\
  c_n & 0 & -c_{n-2} \\
  c_n & -c_{n-1}
\end{bmatrix},
\]

(ii) applying Rayleigh quotient with initial guess \( \sigma_0 = c_n r \) and \( y_0 = [1, 1, \ldots, 1]^T \), and (iii) dividing the final result by \( c_n \). The reason for deferring the division by \( c_n \) is that a root of a polynomial can be extremely sensitive to errors in the coefficients (so it is best not to normalize the polynomial so that its leading coefficient is 1). Use the linear equation solver of part (a) for the iteration. Do at least 5 iterations and declare convergence when the difference between two successive iterates stops decreasing. Input to the routine is an array of polynomial coefficients and the initial guess; the return value is a root. There should also be a boolean global variable \texttt{printIterates} which if \texttt{true} causes printing of the successive iterates starting with the initial guess and ending with the root. Printing should be to full sixteen digit precision. The printed values should, of course, be the raw values divided by \( c_n \).
(c) Apply your rootfinder to the Legendre polynomial of degree 6,

\[ P_6(x) = \frac{1}{16} (231x^6 - 315x^4 + 105x^2 - 5). \]

(Division by 16 is exact in floating-point arithmetic.) First, with `printIterates` set to `true`, apply it for initial guess \( r = 2 \). Then with `printIterates` set to `false`, apply it successively for the values \( r = 0, 0.1, \ldots, 1 \), printing only the root that is found.

Turn in a printed copy of your output and your source code listings.

Function for exchanging values (exchange.m):

```matlab
function [a, b] = exchange(a, b)
temp = a;
a = b;
b = temp;
end
```

Linear equation solver (LinearSystem.m):

```matlab
function [flag, f] = LinearSystem(a, b, d, f)
n = length(f);
c = zeros(1, n-2);
for i = 1:n-1
    if abs(a(i)) < abs(b(i))
        [a(i), b(i)] = exchange(a(i), b(i));
        [d(i), d(i+1)] = exchange(d(i), d(i+1));
        [f(i), f(i+1)] = exchange(f(i), f(i+1));
        if i < n-1
            [c(i), a(i+1)] = exchange(c(i), a(i+1));
        end
    end
end
if a(i) == 0
    flag = -1;
    return;
end
if i < n-1
    a(i+1) = -b(i)/a(i)*c(i) + a(i+1);
end
d(i+1) = -b(i)/a(i)*d(i) + d(i+1);
f(i+1) = -b(i)/a(i)*f(i) + f(i+1);
end
if d(n) == 0
    flag = -1;
    return;
end
f(n) = f(n)/d(n);
f(n-1) = (f(n-1)-d(n-1)*f(n))/a(n-1);
for i = n-2:-1:1
    f(i) = (f(i) - d(i)*f(n) - c(i)*f(i+1))/a(i);
end
```
function root = Rayleigh(c,z,root)
global printInterates;
n = length(c);
cont = 1;
diff = 10000000;
step = 0;
sigma = root * c(n);
if strcmp(printInterates, 'true') == 1
    fprintf(1, 'step	root
');
end
res= 'INF';
while cont == 1
    step = step + 1;
    if strcmp(printInterates, 'true') == 1
        fprintf(1, '%d	%1.15e
', step, sigma/c(n));
    end
    z= z/norm(z);
a = ones(1,n-2) * -sigma;
b = ones(1,n-2) * c(n);
d = -c(1:n-1);
d(n-1) = d(n-1) - sigma;
[flag, z_new]=LinearSystem(a, b, d, z);
if (flag == -1)
    root = sigma/c(n);
    return;
end
    temp= norm(z_new);
    change = (z*z_new')/temp/temp;
    if step <= 5 || abs(change) < diff
        z = z_new;
        sigma = sigma + change;
        diff = abs(change);
    else
        cont = 0;
    end
end
root= sigma / c(n);
return;
end

Polynomial evaluation (eval.m):

function result = eval(c, s)
n = length(c);
result = c(n);
for i = 1:n-1
    result = result*s + c(n-i);
end
return;
end

Execution (test.m):

function test()
global printInterates;
c = [-5, 0, 105, 0, -315, 0, 231]/16;
z = [1, 1, 1, 1, 1, 1];
sigma = 2;
printInterates = 'true';
root = Rayleigh(c, z, sigma);
if strcmp(root, 'INF') == 0
    residual = eval(c, root);
    fprintf(1, '
sigma = %f	root = %1.15e	residual = %1.15e
', sigma, root, residual);
end
printInterates = 'false';
fprintf(1, '
sigma	root	residual
');
for i = 0:10
    sigma = 0.1*i;
    root = Rayleigh(c, z, sigma);
    if root == 'INF'
        fprintf(1, '%f	INF	INF
', sigma);
    else
        residual = eval(c, root);
        fprintf(1, '%f	%1.15e	%1.15e
', sigma, root, residual);
    end
end
end

Output:

step  root
1  2.000000000000000e+000
2  1.189429117020630e+000
3  1.094022652618612e+000
4  9.89911479581516e-001
5  9.42559696194215e-001
6  9.32849058615268e-001
7  9.32470061954809e-001
8  9.32469514204273e-001
9  9.32469514203152e-001
10 9.32469514203152e-001
11 9.32469514203152e-001

4
\[ \sigma = 2.000000 \quad \text{root} = 9.324695142031521 \times 10^{-1} \quad \text{residual} = 2.220446049250313 \times 10^{-16} \]

**Sigma, Root, Residual**

<table>
<thead>
<tr>
<th>Sigma</th>
<th>Root</th>
<th>Residual</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.000000</td>
<td>2.386191860831969 \times 10^{-1}</td>
<td>0.000000000000000 \times 10^{0}</td>
</tr>
<tr>
<td>0.100000</td>
<td>2.386191860831969 \times 10^{-1}</td>
<td>0.000000000000000 \times 10^{0}</td>
</tr>
<tr>
<td>0.200000</td>
<td>2.386191860831969 \times 10^{-1}</td>
<td>0.000000000000000 \times 10^{0}</td>
</tr>
<tr>
<td>0.300000</td>
<td>2.386191860831969 \times 10^{-1}</td>
<td>0.000000000000000 \times 10^{0}</td>
</tr>
<tr>
<td>0.400000</td>
<td>2.386191860831969 \times 10^{-1}</td>
<td>0.000000000000000 \times 10^{0}</td>
</tr>
<tr>
<td>0.500000</td>
<td>6.612093864662644 \times 10^{-1}</td>
<td>4.996003610813204 \times 10^{-16}</td>
</tr>
<tr>
<td>0.600000</td>
<td>6.612093864662645 \times 10^{-1}</td>
<td>2.220446049250313 \times 10^{-16}</td>
</tr>
<tr>
<td>0.700000</td>
<td>6.612093864662645 \times 10^{-1}</td>
<td>2.220446049250313 \times 10^{-16}</td>
</tr>
<tr>
<td>0.800000</td>
<td>6.612093864662645 \times 10^{-1}</td>
<td>2.220446049250313 \times 10^{-16}</td>
</tr>
<tr>
<td>0.900000</td>
<td>9.324695142031522 \times 10^{-1}</td>
<td>1.776356839400251 \times 10^{-15}</td>
</tr>
<tr>
<td>1.000000</td>
<td>9.324695142031521 \times 10^{-1}</td>
<td>2.220446049250313 \times 10^{-16}</td>
</tr>
</tbody>
</table>

2. (4 points) Consider the system of equations

\[
\begin{align*}
5x + 2y + 2z &= 10, \\
x + 2y + z &= 10, \\
x + 5y + 10z &= 10.
\end{align*}
\]

Suppose we are solving this with SOR with relaxation parameter \( \omega = 1.5 \) and have already computed the iterates

<table>
<thead>
<tr>
<th>( k )</th>
<th>( x_k )</th>
<th>( y_k )</th>
<th>( z_k )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>5</td>
<td>-2</td>
</tr>
<tr>
<td>1</td>
<td>0.7</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Determine \( y_1 \).

Solution:

\[
y_1^{GS} = \frac{1}{2} (10 - x_1 - z_0) = \frac{1}{2} (10 - 0.7 - (-2)) = 5.65
\]

\[
y_1 = y_0 + \omega (y_1^{GS} - y_0) = 5 + 1.5(5.65 - 5) = 5.975
\]

3. (5 points) Estimate how many Jacobi iterations would be needed to reduce the error by a factor of \( 10^{-6} \) for \( Ax = b \) where

\[
A = \begin{bmatrix}
4 & 1 & 1 \\
1 & 4 & \ddots \\
& \ddots & \ddots & 1 \\
1 & \ddots & \ddots & 1 \\
\end{bmatrix}
\]

Solution:
\[ B_J = -D^{-1}(L + U) = \begin{bmatrix}
0 & -\frac{1}{4} & 1 \\
-\frac{1}{4} & 0 & \ddots \\
& \ddots & \ddots & -\frac{1}{4} \\
& & -\frac{1}{4} & 0
\end{bmatrix} \]

\[ \rho(B_J) \leq \|B_J\|_{\infty} = \frac{1}{2} \] [also \([1, 1, \ldots, 1]^T\) is an eigenvector for \(-\frac{1}{2}\) and so \(\rho(B_J) = \frac{1}{2}\)]

\[ \left(\frac{1}{2}\right)^m = 10^{-6} \approx 2^{-20} \]

\[ m = 20 \]

4. (5 points) Let

\[ A = \begin{bmatrix}
a_{11} & a_{12} \\
a_{21} & a_{22}
\end{bmatrix}, \quad b = \begin{bmatrix} b_1 \\
b_2
\end{bmatrix}. \]

Plotted below are the level curves (contours) of the function \(\phi(x) = \frac{1}{2}x^TAx - b^Tx\) where \(x = [x_1 \quad x_2]^T\) is variable. Draw the straight line

\[ a_{11}x_1 + a_{12}x_2 = b_1 \]

and the straight line

\[ a_{21}x_1 + a_{22}x_2 = b_2. \]

Explain in words how you constructed these two lines. Recall that relaxing the \(i^{th}\) variable so as to satisfy the \(i^{th}\) equation is equivalent to minimizing \(\phi(x)\) in the direction of the \(i^{th}\) coordinate axis.

The first equation goes through the bottommost and topmost point of each ellipse. The second equation goes through leftmost and rightmost point of each ellipse.
5. (5 points) Show that in the generic minimization method if \( \alpha_i \) is chosen to minimize \( \phi(x_i + \alpha p_i) \), then \( r_{i+1} \) is orthogonal to \( p_i \). Interpret this geometrically.

\[
\begin{align*}
    r_{i+1} &= r_i - \alpha_i A p_i \\
    p_i^T r_{i+1} &= p_i^T r_i - \alpha_i p_i^T A p_i = p_i^T r_i - \frac{p_i^T r_i}{p_i^T A p_i} p_i^T A p_i = 0
\end{align*}
\]

Energy norm of error is minimized on line \( x_i + \alpha p_i \), \( -\infty < \alpha < \infty \) when direction of steepest descent is perpendicular to line.